A Parameterized Algorithm for Exploring Concept Lattices

Peggy Cellier, Sébastien Ferré, Olivier Ridoux, Mireille Ducassé

A Parameterized Algorithm for Exploring Concept Lattices

Peggy Cellier[1], Sébastien Ferré[1], Olivier Ridoux[1] and Mireille Ducassé[2]

[1]IRISA/University of Rennes 1 and [2]IRISA/INSA,
Campus universitaire de Beaulieu, 35042 Rennes, France
firstname.lastname@irisa.fr, http://www.irisa.fr/LIS/

Abstract. Formal Concept Analysis (FCA) is a natural framework for learning from positive and negative examples. Indeed, learning from examples results in sets of frequent concepts whose extent contains only these examples. In terms of association rules, the above learning strategy can be seen as searching the premises of exact rules where the consequence is fixed. In its most classical setting, FCA considers attributes as a non-ordered set. When attributes of the context are ordered, Conceptual Scaling allows the related taxonomy to be taken into account by producing a context completed with all attributes deduced from the taxonomy. The drawback, however, is that concept intents contain redundant information. In this article, we propose a parameterized generalization of a previously proposed algorithm, in order to learn rules in the presence of a taxonomy. The taxonomy is taken into account during the computation so as to remove all redundancies from intents. Simply changing one component, this parameterized algorithm can compute various kinds of concept-based rules. We present instantiations of the parameterized algorithm for learning positive and negative rules.

1 Introduction

Learning from examples is a fruitful approach when it is not possible to a priori design a model. It has been mainly tried for classification purposes [Mit97]. Classes are represented by examples and counter-examples, and a formal model of the classes is learned by a machine.

Formal Concept Analysis (FCA) [GW99] is a natural framework for learning from positive and negative examples [Kuz04]. Indeed, learning from positive examples (respectively negative examples) results in sets of frequent concepts with respect to a minimal support, whose extent contains only positive examples (respectively negative examples). In terms of *association rules* [AIS93,AS94], the above learning strategy can be seen as searching the premises of exact rules where the consequence is fixed. When augmented with statistical indicators like *confidence* and *support* it is possible to extract various kinds of concept-based rules taking into account exceptions [PBTL99,Zak04].

The input of FCA is a formal context that relates objects and attributes. FCA considers attributes as a non-ordered set. When attributes of the context

are ordered, Conceptual Scaling [GW99] allows the attribute taxonomy to be taken into account by producing a context completed with all attributes deduced from the taxonomy. The drawback is that concept intents contain redundant information. In a previous work [CFRD06], we proposed an algorithm based on Bordat's algorithm [Bor86] to find frequent concepts in a context with taxonomy. In that algorithm, the taxonomy is taken into account during the computation so as to remove all redundancies from intents.

There are several kinds of association rules, and several related issues: e.g. find all association rules with respect to some criteria, compute all association rules with a given conclusion or premise. We propose a generic algorithm to address the above issues. It is a parameterized generalization of our previous algorithm. It learns rules and is able to benefit from the presence of a taxonomy. The advantage of taking a taxonomy into account is to reduce the size of the results. For example, the attributes of contexts about Living Things are intrinsically ordered [1]. For a target such as "suckling", a rule such as "Living Things"$\wedge$ "Animalia" $\wedge$ "Chordata" $\wedge$ "Vertebrata" $\wedge$ "Mammalia" $\rightarrow$ "suckling" is less relevant than the equivalent rule "Mammalia" $\rightarrow$ "suckling" where elements redundant with respect to the taxonomy have been eliminated. The presented algorithm can compute various kinds of concept-based rules by simply changing one component. We present two instantiations which find positive and negative rules. Positive rules predict some given target (e.g. predict a mushroom as poisonous), while negative rules predict its opposite (e.g. edible).

The contributions of this article are twofold. Firstly, it formally defines FCA with taxonomy (FCA-Tax) using the Logical Concept Analysis (LCA) framework [FR04], where the taxonomy is taken into account as a specific logic. Secondly, it specifies a generic algorithm which facilitates the exploration of frequent concepts in a context with taxonomy. Quantitative experiments show that taking a taxonomy into account does not introduce slowdowns. Furthermore, the pruning implemented by our algorithm related to the taxonomy can often improve efficiency.

In the following, Section 2 formally defines FCA with taxonomy (FCA-Tax). Section 3 presents the generalization of the algorithm described in [CFRD06] to filter frequent concepts in a formal context with taxonomy. Section 4 shows how to instantiate the algorithm to learn different kind rules. Section 5 discusses experimental results.

2 A Logical Framework for FCA with Taxonomy

In this section, we formally describe Formal Concept Analysis with Taxonomy (FCA-Tax) using the Logical Concept Analysis (LCA [FR04]) framework. We first present an example of context with taxonomy. Then we briefly introduce LCA, concept-based rules, and we instanciate LCA to FCA-Tax. A taxonomy describes how the attributes of the context are ordered and thus a taxonomy is a kind of logic where the subsumption relation represents this order relation.

[1] http://anthro.palomar.edu/animal/table_humans.htm

2.1 Example of Context with a Taxonomy

	Pacific	*Southwest*	*Southeast*	*Region8*	*Hawai*	*Washington*	*Arizona*	*Michigan*	*Mississipi*	*Florida*	*Utah*	*Alaska*	*Oregon*	*California*	*Nevada*	*Endangered*	*Threatened*
Accipitridae										•						•	
Alcedinidae	•															•	
Alcidae						•							•	•			•
Anatidae						•						•					•
Cathartide														•		•	
Charadriidae								•		•				•		•	•
Corvidae					•											•	
Drepanidinae	•															•	
Emberizidae				•													•
Gruidae		•							•							•	
Icteridae			•													•	
Muscicapidae				•													•
Strigidae						•	•				•		•	•			•
Tyrannidae							•				•			•	•	•	
Vireonidae														•		•	

Table 1. *Context of threatened or endangered bird families in the USA*

The context given Table 1 represents the observations of threatened and endangered bird families. Data come from the web site of USFWS[2] (U.S Fish and Wildlife Service). The objects are bird families and each family is described by a set of states and regions where specimens have been observed and by a status: *threatened* or *endangered*. Note that in this context objects are elements of what could be a taxonomy in another context. Figure 1 shows the taxonomy of states and regions of USA.

The context given in Table 1 is the straightforward transcription of the information found on the USFWS site. Note that it is not completed with respect to FCA. For example object *Accipitridae* which has in its description the attribute *Florida* has not the attribute *Southeast*. The information that an object has the attribute *Florida* implies that it has the attribute *Southeast* is implicitly given by the taxonomy. We have observed that this situation is quite common.

2.2 Logical Concept Analysis (LCA)

In LCA the description of an object is a logical formula instead of a set of attributes as in FCA.

[2] http://ecos.fws.gov/tess_public/CriticalHabitat.do?listings=0&nmfs=1

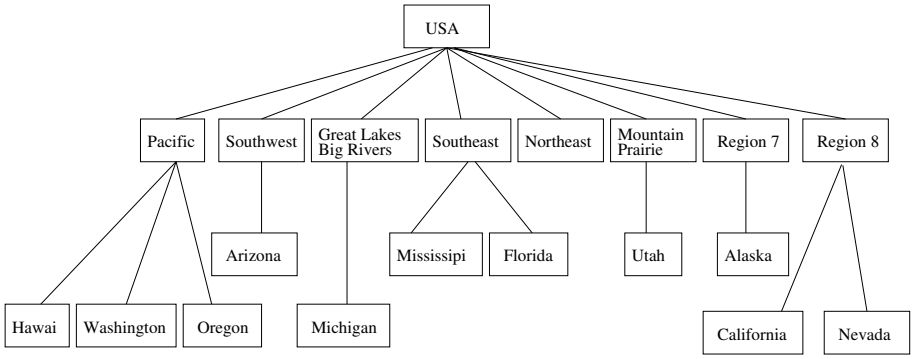


Fig. 1. Taxonomy of USA states and regions.

Definition 1 (logical context). *A* logical context *is a triple* $(\mathcal{O}, \mathcal{L}, d)$ *where* $\mathcal{O}$ *is a set of objects,* $\mathcal{L}$ *is a logic (e.g. proposition calculus) and* d *is a mapping from* $\mathcal{O}$ *to* $\mathcal{L}$ *that describes each object by a formula.*

Definition 2 (logic). *A* logic *is a 6-tuple* $\mathcal{L} = (L, \sqsubseteq, \sqcap, \sqcup, \top, \bot)$ *where*

- L *is the language of formulas,*
- $\sqsubseteq$ *is the subsumption relation,*
- $\sqcap$ *and* $\sqcup$ *are respectively conjunction and disjunction,*
- $\top$ *and* $\bot$ *are respectively tautology and contradiction.*

Definition 3 defines the logical versions of *extent* and *intent*. The extent of a logical formula f is the set of objects in $\mathcal{O}$ whose description is subsumed by f. The intent of a set of objects O is the most precise formula that subsumes all descriptions of objects in O. Definition 4 gives the definition of a *logical concept*.

Definition 3 (extent, intent). *Let* $\mathcal{K} = (\mathcal{O}, \mathcal{L}, d)$ *be a logical context. The definition of extent and intent are:*

- $\forall f \in \mathcal{L} \quad ext(f) = \{ o \in \mathcal{O} \mid d(o) \sqsubseteq f \}$
- $\forall O \subseteq \mathcal{O}\ int(O) = \bigsqcup_{o \in O}\ d(o)$

Definition 4 (logical concept). *Let* $\mathcal{K} = (\mathcal{O}, \mathcal{L}, d)$ *be a logical context. A* logical concept *is a pair* $c = (O, f)$ *where* $O \subseteq \mathcal{O}$*, and* $f \in \mathcal{L}$*, such that* $int(O) \equiv f$ *and* $ext(f) = O$*.* O *is called the extent of the concept* c*, i.e.* ext_c*, and* f *is called its intent, i.e.* int_c*.*

The set of all logical concepts is ordered and forms a *lattice*: let c and c' be two concepts, $c \leq c'$ iff $ext_c \subseteq ext_{c'}$. Note also that $c \leq c'$ iff $int_c \sqsubseteq int_{c'}$. c is called a **subconcept** of c'.

The fact that the definition of a logic is left so abstract makes it possible to accommodate non-standard types of logics. For example, attributes can be valued (e.g., integer intervals, string patterns), and each domain of value can be defined as a logic. The subsumption relation allows to order the terms of a taxonomy.

2.3 Concept-Based Rules

Definition 5 (concept-based rules). *Concept-based rules consider a target $W \in \mathcal{L}$ such that W is a logical formula which represents a set of objects, $ext(W)$, that are positive (respectively negative) examples. Concept-based rules have the form $X \rightarrow W$, where X is the intent of a concept. A rule can have exceptions. These exceptions are measured with statistical indicators like support and confidence, defined below.*

Neither FCA nor LCA take into account the frequency of the concepts they define. An almost empty concept is as interesting as a large one. Learning strategy, however, is based on the generalization of frequent patterns. Thus, statistical indicators must be added to concept analysis. There exists several statistical measures like *support*, *confidence*, *lift* and *conviction* [BMUT97]. In the following, support and confidence are used to measure the relevance of the rules, because they are the most widespread. However, the algorithm presented in this article does not depend on this choice.

Definition 6 (support). *The support of a formula X is the number of objects described by that formula. It is defined as:*

$$sup(X) = \|ext(X)\|$$

where $\|Y\|$ denotes the cardinal of a set Y.

The support of a rule is the number of objects described by both X and W. It is defined as:

$$sup(X \rightarrow W) = \|ext(X) \cap ext(W)\|$$

Definition 7 (confidence). *The confidence of a rule $X \rightarrow W$ describes the probability for objects that are described by X to be also described by W. It is defined as:*

$$conf(X \rightarrow W) = \frac{\|ext(X) \cap ext(W)\|}{\|ext(X)\|}$$

The support applies as well to concepts as to rules. In the case of concept it introduces the notion of *frequent concept* as formalized by Definition 8.

Definition 8 (frequent concept). *A concept is called frequent with respect to a min_sup threshold if $sup(int_c)$ is greater than min_sup.*

2.4 Formal Concept Analysis with Taxonomy

FCA-Tax is more general than FCA because the attributes of the context are ordered. It can be formalized in LCA.

Definition 9 (taxonomy). *A taxonomy is a partially ordered set of terms*

$$TAX =< T, \leq >$$

where T is the set of terms and $\leq$ is the partial ordering. Let x and y be attributes in T, $x \leq y$ means that y is more general than x.

Example 1. In Figure 1, *Hawai* $\leq$ *Pacific* $\leq$ *USA*.

Definition 10 (predecessors, successors, $\mathbf{Min_{tax}}$). *Let $TAX =< T, \leq >$ be a taxonomy and X in T be a set of terms.*
The predecessors *of X in the taxonomy TAX are denoted by*

$$\uparrow_{tax} (X) = \{\, t \in T \mid \exists x \in X : x \leq t \,\}$$

The successors *of X in the taxonomy TAX are denoted by*

$$succ_{tax}(X) = \{\, t \in T \mid \exists x \in X : x > t \wedge (\nexists t' \in T : x > t' > t) \,\}$$

The successors represented by $succ_{tax}(X)$ are immediate ones only. $succ^{+}_{tax}(X)$ is the transitive closure of $succ_{tax}(X)$, i.e. all successors of X in the taxonomy.

$Min_{tax}(X)$ is the set of minimal elements of X with respect to TAX.

$$Min_{tax}(X) = \{\, t \in X \mid \nexists x \in X : x < t \,\}$$

In fact, $Min_{tax}(X)$ is X minus its elements that are redundant with respect to TAX.

Example 2. On the context of Table 1, with the taxonomy of Figure 1, examples of predecessors, successors and Min_{tax} are as follows:

- $\uparrow_{tax} (\{Mississipi, Arizona\}) = \{Mississipi, Southeast, Arizona, Southwest, USA\}$
- $succ_{tax}(\{USA\}) = \{Pacific, Southeast, GreatLakes/BigRivers, Southwest, Northeast, Mountains/Prairie, Region7, Region8\}$
- $Min_{tax}(\{Florida, Southeast, USA\}) = \{Florida\}$

Definition 11 ($\mathcal{L}_{\mathbf{tax}}$). *Let $TAX =< T, \leq >$ be a taxonomy. $\mathcal{L}_{tax} = (L_{tax}, \sqsubseteq_{tax}, \sqcap_{tax}, \sqcup_{tax}, \top_{tax}, \bot_{tax})$ is the logic of FCA-Tax, related to TAX:*

- $L_{tax} = 2^T$ *where T is the set of terms,*
- $\sqsubseteq_{tax}$ *such that $X \sqsubseteq Y$ iff $\uparrow_{tax} (X) \supseteq \uparrow_{tax} (Y)$,*
- $\sqcap_{tax}$ *is* $\cup_{tax}$ *such that* $X \cup_{tax} Y = Min_{tax}(X \cup Y)$,
- $\sqcup_{tax}$ *is* $\cap_{tax}$ *such that* $X \cap_{tax} Y = Min_{tax}(\uparrow_{tax} (X) \cap \uparrow_{tax} (Y))$,
- $\top_{tax} = \{x \in T \mid ext(\{x\}) = \mathcal{O}\}$
- $\bot_{tax} = Min_{tax}(T)$.

$$L = \{\{Hawai\}, \{Oregon\}, \{Pacific\}, \{Southwest\}, ..., \{USA\}, \{Hawai, Southwest\}, ...\} \quad (1)$$

$$\{Michigan, Florida\} \sqsubseteq \{Michigan, Southeast\} \quad (2)$$

$$\{Michigan, Southeast\} \sqcap \{Florida\} = \{Michigan, Florida\} \quad (3)$$

$$\{Mississipi\} \sqcup \{Florida\} = \{Southeast\} \quad (4)$$

$$\top = \{USA\} \quad (5)$$

$$\bot = \{Hawai, Washington, Oregon, Arizona, Michigan, Mississipi, Florida, Utah, Alaska, California, Nevada\} \quad (6)$$

Fig. 2. Examples of operations on the attributes of the taxonomy of Figure 1.

$\sqcap$ and $\sqcup$ follow the usage of description logics, where $\sqcap$ (resp. $\sqcup$) corresponds to intersection (resp. union) over sets of objects.

Example 3. Figure 2 shows examples of operations on the context of bird families. The language, L is the powerset of attributes (1). *Florida* implies *Southeast* in the taxonomy, thus all bird families observed in Michigan and Florida can also be said to have been observed in Michigan and Southeast (2). To be observed in Florida is an information more precise than to be observed in Southeast, thus only the attribute *Florida* is kept in (3). The fact that birds are observed in Michigan or Florida can be generalized by birds which are observed in Southeast (4). The top concept contains only one attribute *USA* (5), and the bottom concept all minimal attributes (6).

All notions defined in LCA apply in FCA-Tax, in particular extent, intent, and concept lattice. Note that the intents of two ordered concepts $c < c'$ differ because one or more attributes are added in $int(c)$, or because an attribute of $int(c')$ is replaced in $int(c)$ by a more specific attribute in the taxonomy, or a combination of both. FCA-Tax differs from FCA in that the intents of concepts in FCA-Tax are without redundancy (see the use of Min_{tax} in the definition of $\sqcup$).

Example 4. $ext(Oregon, California) = \{Alcidae, Strigidae\}$
$int(Alcedinidae, Corvidae) = \{Pacific, Endangered\}$
$sup(Oregon, California \rightarrow Threatened) = 2$
$conf(California \rightarrow Threatened) = 0.5$

Note in the example about the computation of intent that $Corvidae$ has not explicitly the property $Pacific$ but the property $Hawai$. But in the taxonomy the property $Hawai$ implies the property $Pacific$.

3 A Parameterized Algorithm for Finding Concept-Based Rules

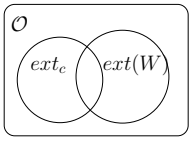
In a previous article [CFRD06] we described an algorithm for finding frequent concepts in a context with taxonomy. The algorithm is a variant of Bordat's algorithm [Bor86] which takes care of the taxonomy for avoiding redundant intents. In this section, a generalization of this algorithm is described in order to search concept-based rules. In a first step, the relation between frequent concepts and rules is presented. In a second step, the parameterized algorithm with this filter function is described.

3.1 From Frequent Concepts to Rules

Relevant rules are rules that are frequently observed. In the general case, see for example [PBTL99], all frequent rules are searched, without any constraint on the premises and conclusions. However, in the learning case, either the premise or the conclusion is fixed by the learning target. So, one searches for frequent rules where conclusions or premises match the target. For instance, learning sufficient conditions for a target W is to search frequent rules like $X \rightarrow W$. For example, in a context describing mushrooms it can be relevant to search properties implying that a mushroom is poisonous, i.e. $X \rightarrow poisonous$.

As explained in Section 2, the frequency of a rule is evaluated by a statistical measure: the support. A rule cannot be more frequent than its premise or its conclusion. Indeed, let c be a concept then $sup(int_c) \geq sup(int_c \cup W) = sup(int_c \rightarrow W)$. This implies that only the intents of frequent concepts are good candidates to be the premises of the searched rules. Therefore, in order to learn rules, the frequent concepts that are computed by the algorithm presented in [CFRD06] need to be filtered. Only most frequent concepts that form relevant rules with respect to statistical measures are kept. Some statistical measures like the support, are monotonous. For instance, given two concepts c and c', $c < c' \Rightarrow sup(c) < sup(c')$ holds. It means that if the support of a concept c is lower than min_sup, all subconcepts of c have a support lower than min_sup. Thus, it is not relevant to explore subconcepts of c.

A filter function, called **FILTER**, is defined in order to take into account these observations. **FILTER** takes two parameters, two sets of objects: ext_c and $ext(W)$. These two parameters are sufficient to compute all statistical measures like support, confidence, lift and conviction as it is illustrated in Figure 3. Using ext_c and $ext(W)$, we can compute $ext_c \cup ext(W)$, $ext_c \cap ext(W)$, and all complements like $\mathcal{O} \setminus ext_c$, $\mathcal{O} \setminus (ext_c \cup ext(W))$. **FILTER**($ext_c$, $ext(W)$) returns two booleans: **KEEP** and **CONTINUE**. **KEEP** tells whether $int_c \rightarrow W$ is a relevant rule with respect to statistical measures. **CONTINUE** allows monotonous properties to be considered. Indeed, **CONTINUE** tells whether there may be some subconcept c' of c such that $int_{c'} \rightarrow W$ is a relevant rule. Thus **FILTER** gives four possibilities: 1) keep the current concept and explore subconcepts, 2) keep the current concept and do not explore subconcepts, 3) do not keep the current concept and explore subconcepts, 4) do not keep the current concept



$$sup(int_c \to W) = \|ext_c \cap ext(W)\|$$

$$conf(int_c \to W) = \frac{\|ext_c \ \cap \ ext(W)\|}{\|ext_c\|}$$

$$lift(int_c \to W) = \frac{\|ext_c \ \cap \ ext(W)\|}{\|ext_c\| \ * \ \|ext(W)\|}$$

$$conv(int_c \to W) = \frac{\|ext_c\| * \|\mathcal{O} \setminus ext(W)\|}{\|ext_c \ \cap \ (\mathcal{O} \setminus ext(W))\|}$$

Fig. 3. Computing measures using the extents of a concept c and a target W.

and do not explore subconcepts. Section 4 illustrates on examples how these four possibilities are used.

3.2 Algorithm Parameterized with Function FILTER

In the first part of this subsection, the data structures used in the algorithm are briefly introduced. In the second part, the algorithm is described. In the last part, the difference with the previous algorithm is given.

The algorithm uses two data structures: $incr_c$ of a concept c and **Exploration**. $incr_c$ contains *increments* of a concept c. Apart from the top concept, each concept s is computed from a concept $p(s)$, called the predecessor of s. Let s be a concept and $p(s)$ be the predecessor of s then there exists a set of attributes, X, such that $ext_s = ext_{p(s)} \cap ext(X)$. We call X an *increment* of $p(s)$, and we say that X *leads* from $p(s)$ to s. All known increments are kept in a data structure $incr_{p(s)}$ which is a mapping from subconcepts to increments: s maps to X iff X leads to s. Notation $incr_{p(s)}[s \mapsto X]$ means that the mapping is overridden so that s maps to X.

Exploration contains frequent subconcepts that are to be explored. In **Exploration**, each concept s to explore is represented by a triple: $(ext_s \mapsto X$, $int_{p(s)}$, $incr_{p(s)})$ where $ext_s = ext(int_{p(s)} \cup X) = ext_{p(s)} \cap ext(X)$ and $\|ext_s\| \geq min\_sup$, which means that X is an increment from $p(s)$ to s.

An invariant for the correction of the algorithm is

$$\forall c \ concept : incr_c \subseteq \{s \mapsto X \mid ext_s = ext_c \cap ext(X) \wedge \|ext_s\| \geq min\_sup\} \ .$$

Thus, all elements of $incr_c$ are frequent subconcepts of c.

An invariant for completeness is

$$\begin{array}{ll} \forall c, s \ concepts : (ext_s \subset ext_c \wedge \|ext_s\| \geq min\_sup \wedge \neg \exists X \subset T : s \mapsto X \in incr_c) \\ \quad \Longrightarrow & (\exists s' \ a \ concept : ext_s \subset ext_{s'} \subset ext_c \wedge \exists X \subset T : s' \mapsto X \in incr_c \ ) \ . \end{array}$$

Thus, all frequent subconcepts of c that are not in $incr_c$ are subconcepts of a subconcept of c which is in $incr_c$.

Algorithm Explore_concepts allows frequent concepts of a context with taxonomy to be filtered with the generic function **FILTER** previously introduced. Some examples of instantiation of the function **FILTER** are given in section 4.

Algorithm 1 Explore_concepts

Require: $\mathcal{K}$, a context with a taxonomy TAX; and min_sup, a minimal support
Ensure: **Solution**, the set of all frequent concepts of $\mathcal{K}$ with respect to min_sup, and **FILTER**

1: **Solution** := $\emptyset$
2: **Exploration**.add($\mathcal{O} \mapsto \top$, $\emptyset$, $\emptyset$)
3: **while Exploration** $\neq \emptyset$ **do**
4: **let** $(ext_s \mapsto X, int_{p(s)}, incr_{p(s)}) = max_{ext}$(**Exploration**) **in**
5: (**KEEP**, **CONTINUE**) := **FILTER**(ext_s, $ext(X)$)
6: **if KEEP** or **CONTINUE then**
7: $\text{int}_s := (\text{int}_{p(s)} \cup_{tax} X) \cup_{tax} \{y \in succ^+_{tax}(X) \mid ext_s \subseteq ext(\{y\})\}$
8: **if CONTINUE then**
9: $\text{incr}_s := \{c \mapsto X \mid \exists c' : c' \mapsto X \in incr_{p(s)} \wedge c = ext_s \cap c' \wedge \|c\| \geq min\_sup\}$
10: **for all** $y \in succ_{tax}(X)$ **do**
11: **let** $c = ext_s \cap ext(\{y\})$ **in**
12: **if** $\|c\| \geq min\_sup$ **then**
13: $incr_s := incr_s[c \mapsto (incr_s(c) \cup \{y\})]$
14: **end if**
15: **end for**
16: **for all** $(ext \mapsto Y)$ in $incr_s$ **do**
17: **Exploration**.add($ext \mapsto Y$, int_s, incr_s)
18: **end for**
19: **end if**
20: **if KEEP then**
21: **Solution**.add(ext_s, int_s)
22: **end if**
23: **end if**
24: **end while**

The concept lattice is explored top-down, starting with the top of the lattice, i.e. the concept labelled by all objects (i.e. the $\top$ concept) (line 2). At each iteration of the while loop, an element of **Exploration** with the largest extent is selected (line 4): $(ext_s \mapsto X, int_{p(s)}, incr_{p(s)})$. This element represents a concept s which is tested with the function **FILTER** (line 5). At line 7, the intent of s is computed by supplementing $(int_{p(s)} \cup_{tax} X)$ with successors of X in the taxonomy. The redundant attributes are eliminated thanks to $\cup_{tax}$.

Then the increment of s are computed; $incr_s$ is computed by exploring the increments of $p(s)$ (step 9) and the successors of the attributes of X in the taxonomy (steps 10-15). Note that increments of $p(s)$ can still be increments of s. For instance, if $p(s) > s$, $p(s) > s'$, and $s > s'$, the increment of $p(s)$ which leads to s' is also an increment of s that leads to s'. If these increments are relevant with respect to **FILTER**, they are added to **Exploration** (lines 16-18).

Finally if s is relevant with respect to **FILTER**, it is added to **Solution** (line 21).

The difference between the algorithm in [CFRD06] and the algorithm presented here is the addition of the function **FILTER** that allows frequent concepts

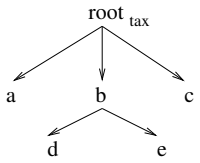


Fig. 4. Taxonomy of the example.

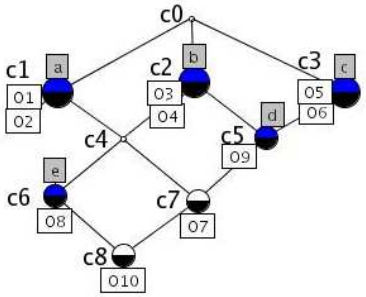


Fig. 5. Concept lattice.

to be filtered and exploration to be stopped. If function **FILTER** always returns (**KEEP** = true, **CONTINUE**=true), then all frequent concepts are eventually computed.

Example 5. Illustration of the computation of **Exploration** is as follows. The taxonomy is presented in Figure 4; Figure 5 shows the explored lattice. We assume min_sup=3. The first 2 steps of computation are explained. Initially, **Exploration** is:

- **Exploration** = $\{(\mathcal{O} \mapsto \top, \emptyset, \emptyset)\}$.

First step: the top of the lattice is explored, i.e. s=c_0. Increments of s are computed from the taxonomy only, as there is no predecessor concept:

- $incr_{c_0}$ = { $\{o_3, o_4, o_7, o_8, o_9, o_{10}\} \mapsto$ {b}, $\{o_1, o_2, o_7, o_8, o_{10}\} \mapsto$ {a}, $\{o_5, o_6, o_7, o_9, o_{10}\} \mapsto$ {c}}
- **Exploration** = $\{(\{o_3, o_4, o_7, o_8, o_9, o_{10}\} \mapsto \{b\}, \emptyset, incr_{c_0}), (\{o_1, o_2, o_7, o_8, o_{10}\} \mapsto \{a\}, \emptyset, incr_{c_0}), (\{o_5, o_6, o_7, o_9, o_{10}\} \mapsto \{c\}, \emptyset, incr_{c_0})\}$.

Second step: an element of **Exploration** with the largest possible extent is explored: $s = c_2$, $p(s) = c_0$. In order to compute $incr_{c_2}$, we have to consider the elements of $incr_{c_0}$ and the elements in the taxonomy.

- $incr_{c_2}$ = { $\{o_7, o_8, o_{10}\} \mapsto$ {a}, $\{o_7, o_9, o_{10}\} \mapsto$ {c, d}, ~~$\{o_5, o_6, o_7, o_9, o_{10}\} \mapsto \{e\}$~~}
- **Exploration** = {($\{o_1, o_2, o_7, o_8, o_{10}\} \mapsto$ {a}, $\emptyset$, $incr_{c_0}$), ($\{o_5, o_6, o_7, o_9, o_{10}\} \mapsto$ {c}, $\emptyset$, $incr_{c_0}$), ($\{o_7, o_9, o_{10}\} \mapsto$ {c,d}, {b}, $incr_{c_2}$), ($\{o_7, o_8, o_{10}\} \mapsto$ {a}, {b}, $incr_{c_2}$)}.

In the second step, attributes d and e are introduced as successors of attributes b, and attributes a and c are introduced as increments of c_0, the predecessor of c_2.

Increment {e} is eliminated because it leads to an unfrequent concept. Attributes c and d are grouped into a single increment because they lead to the same subconcept. This ensures that computed intents are complete.

3.3 Comparison with Bordat's algorithm

The algorithm presented in the previous section is based on Bordat's algorithm. Like in Bordat's version, our algorithm starts by exploring the top concept. Then for each concept s explored, the subconcepts of s are computed; this corresponds to the computation of $incr_s$. Our algorithm uses the same data structure to represent **Solution**, i.e. a trie. The differences between our algorithm and Bordat's are: 1) the strategy of exploration, 2) the data structure of **Exploration** and 3) taking into account the taxonomy.

In our algorithm,

1. we first explore concepts in **Exploration** with the largest extent. Thus, it is a top-down exploration of the concept lattice, where each concept is explored only once. Contrary to Bordat, it avoids to test if a concept has already been found when it is added to **Solution**.
2. Whereas in Bordat's algorithm **Exploration** is represented by a queue that contains subconcepts to explore, in our version the data structure of **Exploration** elements is a triple: $(ext_s \mapsto X,\ int_{p(s)},\ incr_{p(s)})$ where $incr_{p(s)}$ avoids to test all attributes of the context when computing increments of s. Indeed, if an increment X is not relevant for $p(s)$ it cannot be relevant for s, because $ext_s \subset ext_{p(s)}$ and thus $(ext_s \cap ext(X)) \subset (ext_{p(s)} \cap ext(X))$. Thus, the relevant increments of the predecessor of s are stored in $incr_{p(s)}$ and are potential increments of s. In **Exploration**, a concept cannot be represented several times. When a triple representing a concept c is added to **Exploration**, if c is already represented in **Exploration**, the new triple erases the previous one.
3. The attributes can be structured in a taxonomy. This taxonomy is taken into account during the computation of the increments and the intents. Using $\cup_{tax}$ instead of the plain set-theoretic union operation allows computed intents to be without redundancy. Another benefit is that it makes computation more efficient. For instance, when the taxonomy is deep, a lot of attributes can be pruned without testing.

4 Instantiations of the Parameterized Algorithm

In the previous section, the presented algorithm is parameterized with a filter function in order to permit the computation of concept-based rules in FCA-Tax. In this section, we show two instantiations of the filter function to learning. The first one allows *sufficient conditions* to be computed. Sufficient conditions are premises of rules where the conclusion (the target), W, represents the positive examples. In other words, the searched rules are of the form $X \rightarrow W$.

The second one allows *incompatible conditions* to be computed. Incompatible conditions are premises of rules where the conclusion (the target), W, is the negation of the intent of positive examples. In other words, the searched rules are of the form $X \rightarrow \neg W$.

4.1 Computing sufficient conditions

In the generation of sufficient conditions, the learning objective, W, is the target of the rules to be learned. For computing sufficient conditions, one must determine what conjunctions of attributes, X, implies W, i.e. the intents X of concepts such that the rule $X \rightarrow W$ has a support and a confidence greater than the thresholds min_sup and min_conf. Sufficient conditions can be computed with the algorithm described in Section 3 that filters all frequent concepts of a context, by instantiating the filter function with:

$$\begin{aligned}\textbf{FILTER}_{sc}(ext_s, ext(W)) = (\textbf{KEEP} = sup(int_s \rightarrow W) \geq min\_sup \wedge \\ conf(int_s \rightarrow W) \geq min\_conf, \\ \textbf{CONTINUE} = sup(int_s \rightarrow W) \geq min\_sup)\end{aligned}$$

It implies three possible behaviours of the algorithm.

1. The concept s is relevant, i.e. the rule $int_s \rightarrow W$ has a support and a confidence greater than the thresholds. s is a solution and has to be kept, and its subconcepts are potential solutions and have to be explored. Thus $\textbf{FILTER}_{sc}(ext_s)$ returns $(true, true)$.
2. The concept s has a support greater than min_sup but a confidence lower than min_conf. s is not a solution but the confidence is not monotonous and thus subconcepts of s can be solutions. $\textbf{FILTER}_{sc}(ext_s)$ returns $(false, true)$.
3. The concept s has a support lower than min_sup. s is not a solution. As the support is monotonous, i.e. the support of a subconcept of s is lower than the support of s, subconcepts of s cannot be solutions. $\textbf{FILTER}_{sc}(ext_s)$ returns $(false, false)$.

$\textbf{FILTER}_{sc}(ext_s, ext(W))$ can be expressed in terms of extent by simply applying the definitions of support and confidence seen in Section 2. The filter function can therefore be defined by :

$$\begin{aligned}\textbf{FILTER}_{sc}(ext_s, ext(W)) = (\textbf{KEEP} = \|ext_s \cap ext(W)\| \geq min\_sup \wedge \\ \frac{\|ext_s \cap ext(W)\|}{\|ext_s\|} \geq min\_conf, \\ \textbf{CONTINUE} = \|ext_s \cap ext(W)\| \geq min\_sup)\end{aligned}$$

Note that it is easy to use other statistical indicators like *lift* or *conviction*, by adding conditions in the evaluation of **KEEP**.

4.2 Computing incompatible conditions

In the case of incompatible conditions, the learning objective, W, is the negation of the target of the rules to be learned. For computing incompatible conditions, one must determine what conjunctions of attributes, X, implie $\neg W$. In other words, one looks for the intents X of concepts such that the rule $X \rightarrow \neg W$ has a support and a confidence greater than the thresholds min_sup and min_conf.

Incompatible conditions can be computed with the algorithm described in Section 3 by instantiating the filter function with:

$$\begin{aligned}\mathbf{FILTER}_{ic}(ext_s, ext(W)) = (\mathbf{KEEP} = sup(int_s \rightarrow \neg W) \geq min\_sup \wedge \\ conf(int_s \rightarrow \neg W) \geq min\_conf, \\ \mathbf{CONTINUE} = sup(int_s \rightarrow \neg W) \geq min\_sup)\end{aligned}$$

As for sufficient conditions, this can be expressed in terms of extents, using the definitions of support and confidence. The filter function to compute incompatible conditions can thus be defined by :

$$\begin{aligned}\mathbf{FILTER}_{ic}(ext_s, ext(W)) = (\mathbf{KEEP} = \|ext_s \cap (\mathcal{O} \setminus ext(W))\| \geq min\_sup \wedge \\ \frac{\|ext_s \cap (\mathcal{O} \setminus ext(W))\|}{\|ext_s\|} \geq min\_conf, \\ \mathbf{CONTINUE} = \|ext_s \cap (\mathcal{O} \setminus ext(W))\| \geq min\_sup)\end{aligned}$$

5 Experiments

Experimental settings. The algorithm is implemented in CAML (a functional programming language of the ML family) inside the Logic File System LISFS [PR03]. LISFS implements the notion of Logical Information Systems (LIS) as a native Linux file system. Logical Information Systems (LIS) are based on LCA. In LISFS, attributes can be ordered to create a taxonomy (logical ordering). The data structures which are used allow taxonomies to be easily manipulated. For more details see [PR03]. We ran experiments on an Intel(R) Pentium(R) M processor 2.00GHz with Fedora Core release 4, 1GB of main memory.

We tested the parameterized algorithm with two contexts: "Mushroom" and "Java". The Mushroom benchmark[3] contains 8 416 objetcs which are mushrooms. The mushrooms are described by properties such as whether the mushroom is edible or poisonous, the ring number, or the veil color. The context has 127 properties. The Java context[4] taken from [SR06], contains 5 526 objects which are the methods of java.awt. Each method is described by its input and output types, visibility modifiers, exceptions, keywords extracted from its identifiers, and keywords from its comments. The context has 1 624 properties, and yields about 135 000 concepts.

Quantitative Experiments. Many efficient algorithms computing association rules exist[5]. None of them allow a taxonomy to be taken into account. We do not pretend that our algorithm is faster. The objectives of this section are rather to show 1) that our algorithm is reasonably efficient, 2) that given an algorithm

[3] Available at ftp://ftp.ics.uci.edu/pub/machine-learning-databases/mushroom/
[4] Available at http://lfs.irisa.fr/demo-area/awt-source/,
[5] some of them are available on the FIMI web site http://fimi.cs.helsinki.fi/src

min_sup (%)	Number of concepts	Execution time (with taxonomy) (s)	Execution time (without taxonomy) (s)
20	35	4.5	12.2
15	48	4.8	13.2
10	54	5.2	13.6
7.5	86	5.8	14.8
5	189	7.9	16.3
2.5	2300	55.0	54.1

Table 2. Number of concepts and execution times (in seconds) for different values of *min_sup*.

which searches for association rules, the additional mechanisms described in Sections 3 and 4 do not cause any significant slowdown in the presence of a taxonomy, and 3) that they can even improve efficiency in certain cases.

We evaluate our algorithm on the mushroom benchmark, computing frequent concepts. It enables us to compare our algorithm's performance with Pasquier *et al.*'s CLOSE algorithm, using the figures published in Pasquier's PhD thesis [Pas00]. For the same task, execution times are of the same order for both algorithms. For instance, with $min\_sup = 5\%$ CLOSE computation time is about 210s and our algorithm computation time is 290.8s; with $min\_sup = 20\%$ CLOSE computation time is about 50s and our algorithm computation time is only 26.2s. Hence, even without a taxonomy our algorithm does not introduce any slowdown.

On the Java context, we measured the computation time of the frequent concepts, in order to show the impact of the taxonomy on the computation. The taxonomy is derived, for the largest part, from the class inheritance graph, and, for a smaller part, from a taxonomy of visibility modifiers that is predefined in Java. The result is given in Table 2. The third column of that table contains the execution times in seconds of algorithm when the taxonomy is taken into account. The fourth column corresponds to the execution times when the context is a priori completed with the elements of the taxonomy. When $min\_sup = 5\%$, 189 frequent concepts are computed in 8s with taxonomy, whereas they are computed in 16s without taxonomy. When $min\_sup = 2.5\%$, 2300 frequent concepts are computed in 55s with taxonomy, whereas they are computed in 54s without taxonomy. It is explained by the fact that the Java context has few very frequent concepts. Thus pruning using the taxonomy when *min_sup* is greater than 5% occurs very early in the traversal of the concept lattice. For $min\_sup = 2.5\%$, pruning are less impact, computation time is therefore equivalent with or without taxonomy.

Note that from a qualitative point of view, with this context, using the taxonomy allows the number of irrelevant attributes in the intents to be reduced by 39% for $min\_sup = 5\%$.

6 Conclusion

In this article we have proposed an algorithm parameterized by a filter function to explore frequent concepts in a context with taxonomy in order to learn association rules. We have described how to instantiate the filter function in order to find premises of rules where the target are positive examples (sufficient conditions) and negative examples (incompatible conditions). Quantitative experiments have shown that, in practice, taking a taxonomy into account does not negatively impact the performance and can even make the computation more efficient.

The advantage of the presented method is to avoid the redundancies that a taxonomy may introduce in the intents of the frequent concepts.

References

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining associations between sets of items in massive databases. In *Proc. of the ACM-SIGMOD 1993 Int. Conf. on Management of Data*, pages 207–216, May 1993.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499, 12–15 1994.

[BMUT97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In Joan Peckham, editor, *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 255–264. ACM Press, 05 1997.

[Bor86] J. Bordat. Calcul pratique du treillis de Galois d'une correspondance. *Mathématiques, Informatiques et Sciences Humaines*, 24(94):31–47, 1986.

[CFRD06] Peggy Cellier, Sébastien Ferré, Olivier Ridoux, and Mireille Ducassé. An algorithm to find frequent concepts of a formal context with taxonomy. In *Concept Lattices and Their Applications*, 2006.

[FR04] Sébastien Ferré and Olivier Ridoux. An introduction to logical information systems. *Information Processing & Management*, 40(3):383–419, 2004.

[GW99] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, 1999.

[Kuz04] Sergei O. Kuznetsov. Machine learning and formal concept analysis. In *International Conference Formal Concept Analysis*, pages 287–312, 2004.

[Mit97] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Pas00] Nicolas Pasquier. *Data Mining : Algorithmes d'extraction et de réduction des règles d'association dans les bases de données*. Computer science, Université Blaise Pascal - Clermont-Ferrand II, January 2000.

[PBTL99] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. of the 7th Int. Conf. on Database Theory*, pages 398–416. Springer-Verlag, 1999.

[PR03] Yoann Padioleau and Olivier Ridoux. A logic file system. In *Proc. USENIX Annual Technical Conference*, 2003.

[SR06] Benjamin Sigonneau and Olivier Ridoux. Indexation multiple et automatisée de composants logiciels. *Technique et Science Informatiques*, 2006.

[Zak04] Mohammed J. Zaki. Mining non-redundant association rules. *Data Mining Knowl. Discov.*, 9(3):223–248, 2004.